2762
#8
7-28-00
P2.

169.1640

OIPE
MAY 25 2000
JC17
PATENT & TRADEMARK OFFICE

PATENT APPLICATION

RECEIVED
MAY 26 2000
2700 MAIL ROOM

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

| | | |
|---|---|---|
| In re Application of: | ) | |
| | : | Examiner: NYA |
| ALAN TONISSON | ) | |
| | : | Group Art Unit: 2762 |
| Application No.: 09/524,698 | ) | |
| | : | |
| Filed: March 14, 2000 | ) | |
| | : | |
| For: METHOD OF COMPILING | ) | |
| COMPOSITING EXPRESSIONS | : | |
| FOR OPTIMISED RENDERING | ) | May 24, 2000 |

Assistant Commissioner for Patents
Washington, D.C. 20231

CLAIM TO PRIORITY

Sir:

Applicant hereby claims priority under the International Convention and all rights to which he is entitled under 35 U.S.C. § 119 based upon the following Australian priority Application:

PP9237 filed March 16, 1999

A certified copy of the priority document is enclosed.

Applicant's undersigned attorney may be reached in our New York office by telephone at (212) 218-2100. All

This Page Blank (uspto)

correspondence should continue to be directed to our address given below.

Respectfully submitted,

Jack M. Arnold

Attorney for Applicant

Registration No. 25,823

FITZPATRICK, CELLA, HARPER & SCINTO
30 Rockefeller Plaza
New York, New York 10112-3801
Facsimile: (212) 218-2200
75480

This Page Blank (uspto)

09/524.698

AUSTRALIA

OIPE JC17
MAY 25 2000
PATENT & TRADEMARK OFFICE

**Patent Office**
**Canberra**

RECEIVED
MAY 26 2000
TC 2700 MAIL ROOM

I, LEANNE MYNOTT, TEAM LEADER EXAMINATION SUPPORT AND SALES hereby certify that annexed is a true copy of the Provisional specification in connection with Application No. PP 9237 for a patent by CANON KABUSHIKI KAISHA filed on 16 March 1999.

# CERTIFIED COPY OF PRIORITY DOCUMENT

COMMONWEALTH OF AUSTRALIA
AUSTRALIA
PATENT OFFICE

WITNESS my hand this
Seventeenth day of March 2000

[illegible]

LEANNE MYNOTT
TEAM LEADER EXAMINATION
SUPPORT AND SALES

This Page Blank (uspto)

S & F Ref: 442960

ORIGINAL

AUSTRALIA

Patents Act 1990

**PROVISIONAL SPECIFICATION FOR THE INVENTION ENTITLED:**

# Method for Optimising Compilation of Compositing Expressions

| | |
|---|---|
| Name and Address of Applicant: | Canon Kabushiki Kaisha, incorporated in Japan, of 30-2, Shimomaruko 3-chome, Ohta-ku, Tokyo, 146, JAPAN |
| Name of Inventor(s): | Alan Tonisson |

This invention is best described in the following statement:

# METHOD FOR OPTIMISING COMPILATION OF COMPOSITING EXPRESSIONS

## Field of the Invention

The present invention relates to the creation of computer-generated images both in the form of still pictures and video imagery and, in particular, relates to an efficient process and apparatus for creating an image made up by compositing multiple graphical objects.

## Background

Computer generated images are typically made up of many differing components or graphical objects which are composited together and rendered to create a final image. Each graphical object is generally represented ~~as~~ a fill type and a set of outlines. Graphical objects may be rendered by combining them onto a page using compositing operations. An "opacity channel" (also known as a "matte", an "alpha channel", or simply "opacity") is also commonly used as part of the representation of a graphical object. The opacity channel contains information regarding the transparent nature of each element. The opacity channel is typically stored alongside each instance of a colour, so that, for example, a pixel-based image with opacity stores an opacity value as part of the representation of each pixel. In this fashion each graphical object may be interpreted as occupying the entire image area. In this fashion each graphical object may be interpreted as occupying the entire image area, and the outlines of the graphical object can therefore define a region outside of which the object is fully transparent. This region is called the "active region" of the object. A graphical object without explicit opacity channel information is typically understood to be fully opaque within the outlines of the object, and assumed to be completely transparent outside those outlines.

One means for systematically representing an image in terms of its constituent elements and which facilitates in later rendering is an "expression tree". Expression trees typically comprise a plurality of nodes including leaf nodes, unary nodes and binary nodes. Nodes of higher degree, or of alternative definition may also be used. A leaf node, being the outer most node of an expression tree, has no descendent nodes and represents a primitive constituent of an image. Unary nodes represent an operation which modifies the pixel data coming out of the part of the tree below the unary operator. Unary nodes include such operations as colour conversions and convolutions (blurring etc). A binary node typically branches to left and right subtrees, wherein each subtree is itself an expression tree comprising at least one leaf node. Binary nodes represent an operation which combines the pixel data of its two children to form a single result. For example, a binary node may be one of the standard compositing operators such as *over*, *in*, *out*, *atop* and alpha-*XOR*, examples of which and others are seen in Fig. 2. The compositing operations represented by the Porter-Duff compositing operators are arithmetic or bitwise operations including opacity, which are performed on the component values of the colours of a graphical object .

Several of the above types of nodes may be combined to form the expression tree. An example of this is shown in Fig. 1. The result of the left-hand side of the expression tree illustrated in Fig. 1 may be interpreted as an image (Image 1) being colour (Conversion) clipped (In) to spline boundaries (Path). This construct is composited with a second image (Image 2).

Although the active region of a graphical object may of itself be of a certain size, it need not be entirely visible in a final image, or only a portion of the active region may have an effect on the final image. For example, assume an image of a certain size is to be displayed on a display. If the image is positioned so that only the top left corner of the image is displayed by the display device, the remainder of the image is not displayed. The final image as displayed on the display device thus comprises the visible portion of the image, and the invisible portion in such a case need not be rendered.

Another way in which only a portion of the active region of an object may have an effect is when the portion is obscured by another object. For example, a final image to be displayed (or rendered) may comprise one or more opaque graphical objects, some of which obscure other graphical objects. Hence, the obscured objects have no effect on the final image.

Two conventional rendering models include "frame buffer rendering" and "pixel sequential rendering". Conventional frame buffer rendering is based on the "painter's algorithm" which involves rendering multiple objects one at a time into a frame buffer. A compositing operation is carried out for every pixel within the boundaries of an object. A simple frame buffer approach is not sophisticated enough to render arbitrary expressions consisting of compositing operations involving transparency. Extra frame buffers may be required to render arbitrary expressions consisting of compositing operations involving transparency. The extra frame buffers are needed to store intermediate results.

Pixel sequential rendering as opposed to the "object sequential rendering" used in conventional frame buffer based systems involves performing all of the compositing operations required to produce a single pixel before moving on to the next pixel. Interactive frame rates of the order greater than 15 frames per second can be achieved by such conventional methods, because the actual pixel operations are quite simple. Thus, conventional systems are fast enough to produce acceptable frame rates without requiring complex code. However, this is certainly not true in a compositing environment consisting of complex expression trees since the per-pixel cost of compositing is quite high. This is because typically an image is rendered in 24-bit colour in addition to an 8-bit alpha channel, thus giving 32 bits per pixel. Each compositing operator of an expression tree has to deal with each of the four channels.

Problems arise with prior art methods that use the conventional methods. These methods are highly inefficient since the per-pixel cost of evaluating the set of compositing operations required to compute each pixel is too high.

It is an object of the present invention to substantially overcome, or ameliorate, one or more of the deficiencies of the above mentioned methods by the provision of a method for creating an image made up by compositing multiple components.

## Summary of the Invention

According to one aspect of the present invention there is provided a method of creating an image, said image to be formed by compositing at least a plurality graphical objects, each said object having a predetermined outline, said method comprising the steps of: determining an expression for each of a plurality of active regions, each said active region being defined by at least one region outline following at least one of said predetermined outlines or parts thereof; determining expressions representing each of a plurality of effective regions depending on at least one characteristic of at least one active region, wherein each said effective region has a corresponding compositing operation; and applying said corresponding compositing operations substantially to said effective regions to create said image.

According to another aspect of the present invention there is provided a method of creating an image, said image to be formed by compositing at least a plurality graphical objects, each said object having a predetermined outline, said method comprising the steps of: dividing a space in which said outlines are defined into a plurality of regions, each said region being defined by at least one region outline following at least one of said predetermined outlines or parts thereof; determining a plurality of further regions depending on at least one characteristic of at least one region, wherein each said further region has a corresponding compositing operation; mapping said further regions and corresponding compositing operations into a compositing table, comprising a plurality of levels, wherein each said level of said compositing table represents at least one operation for rendering an object or parts thereof or represents an outline for clipping at least one other level; and compositing said image using said compositing table.

According to still another aspect of the present invention there is provided a method of optimising an expression representing the layout of one or more objects, each said object having a predetermined outline, said method comprising the steps of: dividing a space in which said outlines are defined into a plurality of regions, each said region being defined by at least one region outline substantially following at least one of said predetermined outlines or parts thereof; determining a plurality of further regions depending on at least one characteristic of at least one of said regions, wherein each said further region has a corresponding compositing operation; and forming said optimised expression from said plurality of further regions and corresponding compositing operations.

According to still another aspect of the present invention there is provided a method of creating an image, said image to be formed by evaluating a hierarchically structured compositing expression representing said image, said hierarchically structured compositing expression consisting of a plurality of graphical objects and operations arranged as sub-expressions, each said object having a predetermined outline, said method comprising the steps of: determining an active region for each sub-expression of said hierarchically structured compositing expression, said active region being dependent on the predetermined outlines of each graphical object and operations contained in said each sub-expression; calculating an effective region for a corresponding compositing operation of said hierarchically structured compositing expression, depending on at least one characteristic of at least one active region; mapping each said effective region and corresponding operation into a compositing table, comprising a plurality of levels, wherein each said level of said compositing table represents at least one operation for rendering an object or part thereof constituting at least one of said effective regions; and evaluating said hierarchially structured compositing expression using said compositing table.

According to still another aspect of the present invention there is provided a method of calulating effective regions for a plurality of graphical objects, each said object having a predetermined outline, said method comprising the steps of: dividing a space in which said outlines are defined into a plurality of active regions, each said active region being defined by at least one region outline following at least one of said predetermined outlines or parts thereof, each said active region having at least one corresponding Boolean expression; and calculating said effective regions depending on at least one charateristic of at least one corresponding Boolean expression, wherein each said effective region has a corresponding Boolean operation.

According to still another aspect of the present invention there is provided a method of creating an image, said image to be formed by compositing at least a plurality graphical objects, each said object having a predetermined outline, said method comprising the steps of: dividing a space in which said outlines are defined into a plurality of active regions, each said active region being defined by at least one region outline following at least one of said predetermined outlines or parts thereof; determining a plurality of effective regions depending on at least one characteristic of at least one active region, wherein each said effective region has a corresponding compositing operation; mapping said effective regions and corresponding compositing operations into a compositing table, comprising a plurality of levels, wherein each said level of said compositing table represents at least one operation for rendering an object or parts thereof; and compositing said image using said compositing table.

## Brief Description of the Drawings

Embodiments of the invention are described with reference to the drawings, in which:

Fig. 1 shows a compositing containing a number of overlapping objects;

Fig. 2 shows the result of a variety of compositing operators useful with the present invention;

Fig. 3 shows an expression tree with its corresponding level activation table which is formed using natural mapping methods;

Fig. 4 is a flow diagram of a method of compiling an optimal level activation table in accordance with the preferred embodiment of the invention;

Fig. 5 shows an expression tree in which the expression for the active region is shown next to each level in accordance with the preferred embodiment of the present invention;

Fig. 6 shows the expression tree of Fig. 5 in which the expression for the clipped active region is shown next to each level in accordance with the preferred embodiment of the present invention;

Fig. 7 shows the expression tree of Fig. 5 in which the expression for the effective region is shown next to each level in accordance with the preferred embodiment of the present invention;

Fig. 8 shows an optimal level activation table for the expression tree of Fig. 5, in accordance with the preferred embodiment of the present invention;

Fig. 9 shows an apparatus upon which the preferred embodiment can be implemented;

Tables. 1 and 2 show level activation table entries implementing compositing operations if the active regions of the objects are not considered;

Table. 3 shows the rules for calculating the active region of any compositing expression in accordance with the preferred embodiment;

Table. 4 shows the rules for compiling compositing expressions, into level activation tables, whose primary operation has a primitive left operand;

Table. 5 shows some rules for compiling compositing expressions, into level activation tables, whose primary operation has a complex left operand;

Table. 6 shows a condensed version of Table. 5 combined with Table. 3 refined; and

Appendix. A gives a pseudocode listing of an implementation in accordance with the preferred embodiment of the present invention.

## Detailed Description

Where reference is made in any one or more of the drawings to steps and/or features, which have the same reference numerals, those steps and/or features are for the purposes of the description the same, unless the contrary appears.

The preferred embodiment is a method of compiling a table of levels known as a "level activation table" which is used to optimally evaluate a given compositing expression such that the number of pixel operations needed to evaluate the expression is minimised. The preferred embodiment is preferably used in a compositing model in which objects are composited onto a page using a stack machine and an associated compositing stack. The stack may be manipulated using standard stack operations such as *push*, *pull* and *pop*. Instructions for the stack machine are taken from the level activation table. An example of a level activation table 300 is illustrated in Fig. 3. Each level normally represents a graphical object plus a graphics operation which controls the compositing of the objects associated fill data with the data from exposed levels beneath it. Compositing of the level activation table proceeds from the lowest exposed level, up to the highest. At each level, the operation specified in the level activation table is performed, and the result is pushed onto the compositing stack. The preferred embodiment does not transform expressions. However, expressions can be simplified and pre-processed to minimise resource usage before applying the method of the preferred embodiment.

Fig. 4 is a flow diagram of a method of compiling an optimal level activation table to optimally evaluate a given compositing expression in accordance with the preferred embodiment of the invention. The method commences at step 402, where a check is carried out to determine whether the given compositing expression is "primitive" (ie: represented by one "level" of the form *push A*). The processing continues at step 404, where if the expression is not primitive but is "complex" (ie: represented by more than one active region), then a check is carried out to determine if the left operand of the expression is primitive. In the preferred compositing model, the left operand (ie: the source fill data) for an operation is taken from the current level and the right operand (ie: the destination fill data) is popped from the top level of the stack. At the next step 406, if the left operand of the expression is not primitive, then the effective region of the left operand is calculated using level activation table compilation rules for complex left operands (see Table. 5 attached) in accordance with the preferred embodiment. The effective region and the level activation table compilation rules will be described in more detail later in this document. The processing continues at step 408, where a "*pop*" level is added to the optimal level activation table of the given compositing expression. The *pop* level is a level that represents an operation to pop its left operand from the stack, during the compositing of the optimal level activation table, rather than explicitly being part of the level. At the next step 410, a check is carried out to determine if the effective region is "complex". Complex, in this context, means any effective region that involves more than one object boundary. If the effective region is complex, then clipping levels, and duplicate levels if necessary, are added to the optimal level activation table of the compositing expression at step 412. Duplicate levels only need to be added if the effective region for the operation is complex. In addition, if the disjunctive form of the Boolean expression for the clipping region involves any unions, the level that is being clipped will need to be duplicated. The clipping region of the left operand is the minimum region in which the left operand needs to be evaluated in order to be able to evaluate the compositing expression in which it is contained. Therefore, the clipping levels are added to the optimal level activation table to avoid unnecessary pixel operations during compositing. Clipping regions and clipping levels will be further explained later in this document. The processing continues at step 414, where if no clipping levels need to be added or after the necessary clipping and duplicate levels have been added to the optimal level activation table, the method outlined by the flowchart is repeated in order to recursively compile the levels of the level activation table for the left operand of the left operand of the given compositing expression.

If the left operand is primitive at step 404, then processing proceeds to step 416 where the left operand's effective region is calculated using the level activation table compilation rules for primitive left operands (see Table 4 below) in accordance with the preferred embodiment. At the next step 418, the appropriate *op* value level is added for the left operand. As with *pop* and *clip* levels, duplicate *op* value levels can be added at this point to enable the compositing of a union of primitives or their complements. The processing continues at step 420, where a check is carried out to determine if the effective region of the left operand is complex. If the effective region is complex, then clipping levels, and necessary duplicate levels are added to the left operand of the compositing expression at step 422. The processing continues at step 424, where if no clipping levels need to be added or after the necessary clipping and duplicate levels have been added, a check is carried out on the effective region of the left operand to determine whether a "*push*" level needs to be added. The *push* level represents the case where the operation produces values that are equal to the left operand in the region where the right operand is missing. The *push* level increases the depth of the compositing stack. Duplicate push levels can also be added at this point if the effective region for the operation is complex. If required, the *push* levels are added to the level activation table, at the next step 426. Otherwise, the processing continues to step

440 where the right hand operand of the compositing expression is recursively compiled using the same method as for the left-hand operand.

At step 428 which follows step 426, a check is carried out to determine if the effective region of the left operand is complex. If the effective region is complex, then clipping levels, and necessary duplicate levels, are added to the level activation table of the compositing expression, at step 430. Otherwise, the processing continues at step 440 where the levels for the right-hand operand of the compositing expression are recursively compiled using the same method as for the left-hand operand. Once the levels for the right-hand operand of the compositing expression have been compiled, the optimal level activation table is complete and the processing terminates at step 442.

If at step 402, the given compositing expression is primitive, then processing continues to step 432 where the effective region of the given expression is calculated as the intersection of the active and clip regions of the given expression. At the next step 434, a "*push* value" level is added to the expression. Again, duplicate push levels can also be added at this point if the effective region for the operation is complex. The processing continues at step 436, where a check is carried out to determine if the effective region of the expression which resulted from the added *push* level is complex. If the effective region is complex, then clipping levels are added to the left operand of the compositing expression at step 438. Otherwise, the processing terminates at step 442.

Once the optimal level activation table has been compiled using the method of the preferred embodiment, the stack machine of the preferred compositing model can begin compositing the expression represented by the level activation table. As discussed above, compositing of the optimal level activation table proceeds from the lowest level, up to the highest. At each level, the operation specified in the optimal level activation table is performed, and the result is pushed onto the compositing stack. It is has been found that there is a trade-off between the number of levels used and the number of pixel operations performed to evaluate the compositing expression. In some instances, the number of pixel operations performed can be reduced by introducing more levels.

The evaluation of the regions and methods of the preferred embodiment will now be discussed in more detail. For an explanation of mapping compositing expressions onto level activation tables, reference is made to the following example:

<u>Example</u>

As seen in Fig. 3, for a given compositing expression 310 consisting of graphical objects and compositing operations, a natural mapping, obtained by evaluating the expression tree from right to left, produces a level activation table

300 from which the expression can be evaluated. The natural mapping of the level activation table occurs in four stages as follows:

1. Write the expression in Polish notation.

   i.e. *over over in A B out C D E*

2. Combine each operand that is adjacent to a graphic operand to form a level.

   i.e. *over over (in A) B (out C) D E*

3. Replace each remaining graphic with a level that pushes pixels from the graphic onto the stack.

   i.e *over over (in A) (push B) (out C) (push D) (push E)*

4. Replace each remaining operator with a level that pops its source off the stack.

   i.e. *(over pop) (over pop) (in A) (push B) (out C) (push D) (push E)*

In the example,

| | |
|---|---|
| '(*push X*)' | denotes a level representing an operation where a colour value defined by fill X is pushed onto the stack. In this type of level, the left operand is explicitly designated by the level. |
| '(*op X*)' | denotes a current level with fill *X* and a colour operation *op*. The source fill data *X* for the operation is taken from the current level and the destination fill data is popped from the top level of the stack. |
| '(*op* pop)' | denotes a level representing an operation *op* where a colour value defined by fill X is popped from the stack. In this type of level, the left operand is explicitly designated by the level. |

The result of the natural mapping of the compositing expression 310 is the level activation table 300 which looks like a flattened expression tree where each sub-expression corresponds to a contiguous subset of the table 300. The same table entries can also be calculated by an algorithm that recursively processes each sub-expression in a single pass through the expression tree. Expressions can be processed either right to left or left to right, depending on the order in which the levels need to be generated. The expression tree 310 will be evaluated correctly inside the intersection of the active regions of all sub-expressions using the level activation table 300. However, for pixels outside that region, there is no guarantee that the preferred compositing model will calculate the correct value using the level activation table 300 generated by the natural mapping. Therefore, extra work is needed to produce correct optimal level activation tables.

The method of the preferred embodiment uses active region analysis, to avoid redundant pixel operations by clipping operations to the regions in which both of their operands are available. The method can also clip operands to the region in which the operations that will be applied to them produce non-transparent pixels. In most instances, region analysis is necessary to produce correct level activation tables as will be discussed below. The method of the preferred embodiment can evaluate any given compositing expression provided that there are sufficient levels available to represent the expression, and sufficient stack space to evaluate the expression.

Performing unnecessary compositing operations can be avoided by only performing an operation when its level is active; i.e. the operation is only performed for pixels that lie inside the active region of the graphical object corresponding to the level. This behaviour is both efficient and close to correct, but as discussed above it does not always produce the correct result for all compositing expressions. Unfortunately, the level activation tables generated using the natural mapping discussed above will only produce the correct pixel values under special circumstances or for special types of expressions.

The preferred compositing model described above works best with an expression tree that leans to the right. Ideally, each compositing operation corresponds to a single level in the level activation table. In this situation, each level in the table refers to an object plus an operation to combine the object's pixels with the top of the compositing stack. In this case, the number of pixels on the compositing stack will never be more than one.

If the expression tree leans entirely to the right and the expression contains no *in, rin, out* or *ratop* operations, the resulting level activation table will correctly compute the value of the expression for every pixel on the page. If the expression does not fall into this category, careful consideration of the active region of each sub-expression is required to understand how to make the compositing model calculate the correct pixel value. The necessary modifications to the level activation table will involve adding extra clipping levels and/or replacing levels with sets of levels which compute different operations in different regions of the page.

If a compositing operation is performed when the stack is empty, it is known that this can be compensated for by using an opaque white colour value as the right operand. However, in such a case the preferred embodiment adds one or more extra levels to the level activation table that each represent a push of the background colour onto the stack. *Over* operation levels are also required to render the expression correctly.

The problem is that the decision to not do any processing needs to take into account the active regions of both of the operands. The preferred compositing model either applies the operation everywhere on the page or only inside the left operand's active region. There are several situations where incorrect results will be produced with the simple natural mapping of expressions onto level activation tables as described above:

1. In the case of a level that takes its source from the fill table, the operation is applied inside the boundaries of the source operand. This means that a stack operation is sometimes applied when the pixels of the destination operand have not been placed on the stack by levels below.

2. The results of some compositing operations affect pixels that are outside the active region of the source operand, so levels which take their source pixels from the fill table will not always be active in the correct area. For example, *rin* makes pixels that lie outside the source operand's active region transparent.

3. In the case of a level which *pop*s its source off the stack, the operation is applied everywhere regardless of whether its operands have actually been placed on the stack or not by lower levels.

The simplest way to solve these problems is to replace each primitive sub-expression, *A*, in the compositing expression with *A over glass*, where *glass* represents a transparent fill the size of the whole page. If an expression of this form is converted to a level activation table via the natural mapping, the stack depth at any given level will be the same for every pixel in the page. The resulting table will always compute the correct value for every pixel, but when using this approach, every level except the newly added *over* levels will be active for the entire page. This results in more pixel operations being performed than necessary.

In order to produce a more efficient solution, it is necessary to pay careful attention to the region in which each sub-expression is active. Ideally, each level should only perform a stack operation inside the smallest region outside of which it can be determined (by region analysis) that the resulting pixels will be transparent. Operations only need be performed in the region in which both operands are active.

Tables 1 and 2, attached, show level activation table entries implementing compositing operations if the active regions of the objects are not considered. Table 1 illustrates problems 1 and 2 for each compositing operation, and Table 2 covers problem 3. The "Operation" column in each table contains an expression to be evaluated by applying a compositing operation. The "Region" column lists all combinations of the active regions of the operands and the "Value" column gives the correct value of the compositing expression for each combination. The "Level" column indicates the level that is used to implement the operation. Table 1 shows operations where the source is taken from the fill table. Table 2 shows operations that take (ie: *pop*) their source off the stack.

In Table 1, the “Stack” column shows the pixel on top of the stack. The “Result” columns indicate the value that the level will calculate. The operations in Table 1 do not change the stack depth, they replace the destination pixel on top of the stack with the result shown in the "Result" column. In Table 2, the “Stack” column shows the values of the top two pixels on the stack (the left one is the top of the stack). The operations in Table 2 always replace the top two values on the stack with the result of the operation shown in the "Result" column, thus reducing the stack depth by 1.

An expression containing a question mark indicates that an unknown value from the top of stack is used. If the stack is not empty, this results in the stack depth being incorrectly reduced and the wrong value being pushed onto the stack. If the stack is empty, this is not a problem since the compositing model will substitute an opaque white pixel for the operand. A question mark by itself indicates that the top of stack is left unchanged. It is assumed that the part of the table responsible for computing each operand pushes exactly one pixel onto the stack inside the active region of the sub-expression and does not push anything outside the active region.

The “Correct” columns shows the operation that needs to be performed in order to leave the correct value on the stack. The result in the "Correct" columns take into account the active regions. However, they do not take into account any clipping and in some cases a *pop* is required to remove an unused pixel that is outside the active region of the expression. The operand *pop* indicates that the operand is the top of stack, which is popped off the stack before pushing the result onto the stack. Further optimisations described below will add extra clipping levels to avoid generating the pixels that need to be popped. In Table 2 the operations in the “Correct” column are assumed to pop both of their operands off the stack before pushing the result onto the stack.

Note that in some cases, a *push* operation replaces the compositing operation. Where a *push* appears, the increase in stack depth compensates for the fact that the destination operand is assumed to be missing from the stack.

The active region of a graphical object was defined above as a region outside of which the object is known to be transparent. The following paragraphs describe exactly what the active region of an expression is and how to calculate it.

The definition of active regions can be extended to arbitrary compositing expressions by defining the active region of an expression to be the smallest region outside of which the expression is known to evaluate to *glass*.

The active region of any compositing expression can be calculated by applying some rules, as illustrated in Table 3 below, to combine active regions of sub-expressions. For each compositing operation, the active region of the result of the operation can be defined in terms of the active regions of its operands. For example, the active region of *A over B* is the union of the active regions of *A* and *B*.

The active region of an expression can be calculated by recursively applying the rules shown in Table 3 to each sub-expression. These rules are applied from the bottom of the tree upward towards the root. Fig. 5 illustrates an expression tree 500 in which the expression for the active region is shown next to each level.

Note that in some cases, the active region of a sub-expression may not be contained in the active region of an expression in which it appears. In these cases, the sub-expression will generate pixels that will not be needed higher up the expression tree. To avoid unnecessary pixel operations each sub-expression needs to be clipped to the intersection of the active regions of each expression in which it appears. Therefore, the clipping region of a sub-expression can be defined to be the intersection of the active regions of all of the (sub) expressions in which it appears. As discussed above, the clipping region of a sub-expression is the minimum region in which the sub-expression needs to be evaluated in order to be able to evaluate the expression in which it is contained.

Clipping regions are calculated downward from the root of the expression tree towards the leaves. The clipping region for each sub-expression in a compositing expression can be calculated using the follow steps:

1. Label each sub-expression with its active region;
2. Set the clipping region for the top-level expression to be its active region; and
3. Recursively set the clipping region of each sub-expression to the intersection of its active region and its parent's clipping region.

Figure 6 illustrates an expression tree 600 whereby the root node of each sub-expression in the expression tree 600 is labelled with the clipping region of the sub-expression. For example, the clipping region of the sub-expression *D out E* is *D*.

The clipping region is the smallest region in which each sub-expression needs to be evaluated. However, to evaluate expressions correctly, each operation should only be applied in the region inside which both of its operands are available, i.e. in the intersection of their clipping regions. This region is called the effective region of the operation. Note that the effective region of an operation is generally not the same as the clipping region of the sub-expression that the operation forms the root thereof. The effective region is usually a proper subset of the clipping region of the sub-expression that it is the root of. Figure 7 illustrates an expression tree 700 whereby the root node of each sub-expression in the expression tree 700 is labelled with the effective region of the sub-expression. For example, the effective region of the sub-expression *D out E* is $D \cap E$. Leaf nodes can be considered to be *push* operations whose effective regions are the same as their clipping regions.

The effective region of each sub-expression of an expression tree is used in the method of the preferred embodiment to compile compositing expressions into an optimal level activation table. Unfortunately, effective regions can be arbitrarily complex Boolean expressions which are not supported by the preferred compositing model. This problem can be overcome because every boolean expression can be rewritten in disjunctive form, as a disjoint union of intersections of primitives or the complements of primitives. Effective regions in disjunctive form can be compiled into level activation tables that will perform the necessary calculations.

Fig. 8 shows an optimal level activation table for the expression 700, of Fig. 7 written in its disjunctive normal form 710. All of the effective regions except the *atop* and its argument are simple intersections of primitive regions or their complements. The *atop* and its argument need to be clipped to $(A \cap (B \cap C) \cup D)$ which can be rewritten as $(A \cap B \cap C \cap \bar{D}) \cup (A \cap D)$.

The sub-expression `D out E` of Fig. 7 is represented by three levels in the table 800. It is represented by the `push E` level 810, the `out D` level 820 and the `push D` level 840. The extra `push D` is needed because the *out* operation will not generate correct results outside $D \cap E$. Whenever a primitive is the left operand of an operation with the same effective region, the primitive can be combined with the operation into a single level. If not, there needs to be an extra `push` level, which is active only outside the effective region of the operation. This ensures that the `push` is never active at the same time as the operation that it replaces. In all other cases in the example, the clipping region of each left primitive is the same as the operation that composites it onto the page, so the `push` can be combined with the operation.

As seen in Fig. 8, the `atop A` level is duplicated as levels 850 and 860. As discussed above in connection with Fig. 4, this is to allow a single level to be clipped to a union of primitives or their complements. The first `atop A` 860 is active in the region $A \cap D$ and the second 850 is active in $A \cap B \cap C \cap \bar{D}$. Since the effective regions of the two levels are disjoint, at most one of them is active at any one time. Together, the two levels perform the operation in the region $(A \cap B \cap C \cap \bar{D}) \cup (A \cap D)$.

Tables 4 and 5 summarise the rules for compiling compositing expressions into level activation tables according to the preferred embodiment. These tables are derived by combining information from Tables 1, 2, and 3. All expressions fall into three categories:

1. Primitive expressions: represented by one level of the form *push A;*

2. Expressions whose primary operation has a primitive left operand: Table 4 describes the levels needed to evaluate these; and

3. Expressions whose primary operation has a complex left operand: Table 5 describes the levels needed to evaluate these.

The "Levels" columns of Tables 4 and 5 show the levels needed to implement the operation for a sub-expression. The "Clip" column shows the region that each level needs to be clipped to. The region denoted by "*clip*" in the "Clip" column is the region that is passed down recursively to the sub-expression. In the "Clip" column "*A*" and "*B*" denote the active regions of the sub-expressions *A* and *B*. For example, for a sub-expression of the form *A out B* clipped to a region *clip*, where *A* is a primitive, Table 4 shows that the sub-expression can be evaluated using a sub-table to implement the sub-expression *B* and two levels (*push A* and *out A*) to implement the out operation. The sub-table is represented by "*table B*" in the "Levels" column of Table 4. The *push A* level should only be active in the intersection of *clip* and the complement of the active region of *B*. The *out A* level should only be active in the intersection of *clip* and the active region of *B*. The sub-table that implements *B* is generated by applying the same rules recursively with a clip region equal to the intersection of *clip* and the active region of *A*.

The region analysis described above did not take into account any knowledge about the opaqueness of objects. If it is known that some of the objects are fully opaque, further optimisations become possible. The opacity channel of each object in an expression can be checked to determine the transparent nature of each object. This check can be carried out prior to the step of calculating the regions in the method of the preferred embodiment.

The opaque region of an expression can be defined to be the region in which it is known to be fully opaque. The opaque region of an expression is always a subset of its active region. If the opaque regions of the sub-expressions of an expression are known, the active regions in the expression can be further refined. Opaque regions can be calculated in a similar way to active regions. Table 6 gives rules for calculating opaque regions of complex expressions, and an updated set of rules for calculating clipping regions and active regions. Table 6 is a condensed version of Table 5 combined with Table 3 refined to take into account opaque regions. The opaque regions of *A* and *B* are denoted: $O_A$ and $O_B$ respectively. The "Clip A" and "Clip B" columns give the clipping regions of the sub-expressions *A* and *B*. The parent expression's clipping region is not explicitly shown, but it must be included when calculating clipping regions of a sub-expression.

As seen in Table 6, the active regions are smaller in some cases when it is known that one or both of the operands are opaque or partially opaque. In cases when an operand is fully opaque, performing a pixel operation can be avoided altogether. For example, to evaluate *A over B*, when *A* is opaque, the effective region is $A \cap (B \cap \bar{O}_A) = A \cap B \cap \bar{A} = \varnothing$.

Appendix A gives a pseudocode implementation of a compiler for compositing expressions which produces optimal level activation tables in accordance with the preferred embodiment. The pseudocode syntax is close to C++ while avoiding use of C++ constructs. The compiler is not intended to be efficient. It's purpose is to illustrate how to compile compositing operations. A real implementation would preferably not be written recursively. Further, the compiler seen in Appendix A does not take into account opaque regions. Finally the pseudocode implementation assumes that several abstract data types are available: *Expression, Operation, Region, Table,* and *Level.* These represent compositing expressions, compositing operations, Boolean expressions representing regions of the page, level activation tables, and individual levels in a level activation table, respectively.

**Preferred Embodiment of Apparatus(s)**

The preferred embodiment of the present invention is described as a computer application program hosted on the Windows™ operating system developed by Microsoft Corporation. However, those skilled in the art will recognise that the described embodiment may can be implemented on computer systems hosted by other operating systems. For example, the preferred embodiment can be performed on computer systems running UNIX™, OS/2™, DOS™. The application program has a user interface which includes menu items and controls that respond to mouse and keyboard operations. The application program has the ability to transmit data to one or more printers either directly connected to a host computer or accessed over a network. The application program also has the ability to transmit and receive data to a connected digital communications network (for example the "Internet").

The preferred embodiment of the invention can be practised using a conventional general-purpose (host) computer system, such as the computer system 900 shown in Fig. 9, wherein the application program discussed above and to be described with reference to the other drawings is implemented as software executed on the computer system 900. The computer system 900 comprises a computer module 910, input devices such as a keyboard 920 and mouse 930, output devices including a printer 940 and a display device 950. A Modulator-Demodulator (Modem) transceiver device 960 is used by the computer module 910 for communicating to and from a communications network, for example connectable via a telephone line or other functional medium. The modem 960 can be used to obtain access to the Internet, and other network systems.

The computer module 910 typically includes at least one processor unit 980, a memory unit 990, for example formed from semiconductor random access memory (RAM) and read only memory (ROM), input/output (I/O) interfaces including a video interface 975, and an I/O interface 985 for the keyboard 920 a mouse 930 and optionally a joystick (not illustrated). A storage device 995 is provided and typically includes a hard disk drive 955 and a floppy disk drive 945. A CD-ROM drive 935 is typically provided as a non-volatile source of data. The components of the computer module 910, typically communicate via an interconnected bus 915 and in a manner which results in a conventional mode of operation of the computer system 900 known to those in the relevant art. Examples of computers on which the embodiments can be practised include IBM-PC's and compatibles, Sun Sparcstations or alike computer systems evolved therefrom. Typically, the application program of the preferred embodiment is resident on a hard disk drive 955 and read and controlled using the processor 980. Intermediate storage of the program and the print list and any data fetched from the network may be accomplished using the semiconductor memory 925, possibly in concert with the hard disk drive 955. In some instances, the application program may be supplied to the user encoded on a CD-ROM or floppy disk, or alternatively could be read by the user from the network via the modem device 970.

The method of the preferred embodiment can alternatively be implemented in dedicated hardware such as one or more integrated circuits performing the functions or sub functions of the steps of the method. Such dedicated hardware can include graphic processors, digital signal processors, or one or more microprocessors and associated memories.

The foregoing only describes a small number of embodiments of the present invention, however, modifications and/or changes can be made thereto by a person skilled in the art without departing from the scope and spirit of the invention.

For example, the method of the preferred embodiment could be used in a system that uses a data flow processor rather than a stack machine to composite graphic object data.

Further, the method described above is not limited to the compositing of graphics objects. Any process involving the evaluation of expressions involving .

**Aspects of the Invention:**

1. A method of creating an image, said image to be formed by compositing at least a plurality graphical objects, each said object having a predetermined outline, said method comprising the steps of:

determining an expression for each of a plurality of active regions, each said active region being defined by at least one region outline following at least one of said predetermined outlines or parts thereof;

determining expressions representing each of a plurality of effective regions depending on at least one characteristic of at least one active region, wherein each said effective region has a corresponding compositing operation; and

applying said corresponding compositing operations substantially to said effective regions to create said image.

2. A method of creating an image, said image to be formed by compositing at least a plurality graphical objects, each said object having a predetermined outline, said method comprising the steps of:

dividing a space in which said outlines are defined into a plurality of regions, each said region being defined by at least one region outline following at least one of said predetermined outlines or parts thereof;

determining a plurality of further regions depending on at least one characteristic of at least one region, wherein each said further region has a corresponding compositing operation;

mapping said further regions and corresponding compositing operations into a compositing table, comprising a plurality of levels, wherein each said level of said compositing table represents at least one operation for rendering an object or parts thereof or represents an outline for clipping at least one other level; and

compositing said image using said compositing table.

3. A method according to paragraph 2, wherein a further region is determined on the basis that a particular region corresponds to a primitive expression.

4. A method according to paragraph 1, wherein an effective region is determined on the basis that a particular active region corresponds to a primitive expression.

5. A method according to paragraph 3, wherein said further region is an effective region.

6. A method according to paragraph 5, wherein said effective region is equal to the intersection of the clip and active regions of said particular corresponding compositing expression.

7. A method according to any one of paragraphs 3 to 6, wherein a level comprising a push operation is added to said compositing table.

8. A method according to any one of paragraphs 3 to 7, wherein a corresponding compositing expression of said further region is complex.

9. A method according to paragraph 8, wherein a level comprising a clip operation is added to said compositing table.

10. A method according to any one of the preceding paragraphs, wherein a further region is determined on the basis that said corresponding compositing operation has a complex left operand.

11. A method according to paragraph 10, wherein a level comprising a pop operation is added to said compositing table.

12. A method according to paragraph 11, wherein said pop operation will remove any unused pixel being outside a further region representing said complex left operand, during compositing of said complex left operand.

13. A method according to paragraph 12, wherein said further region is the active region of said complex left operand.

14. A method according to paragraph 12, wherein said further region is transformed to a still further region by said pop operation.

15. A method according to paragraph 14, wherein said still further region is the effective region of said complex left operand.

16. A method according to paragraph 15, wherein said still further region corresponds to a complex expression.

17. A method according to paragraph 16, wherein a level comprising a clip operation is added to said compositing table.

18. A method according to any one of the preceding paragraphs, wherein a further region is determined on the basis that said corresponding compositing operation has a primitive left operand.

19. A method according to paragraph 2, wherein a level comprising an operation and a data fill value of a particular object constituting said further region, is added to said compositing table.

20. A method according any one of paragraphs 18 or 19, wherein said further region corresponds to a complex expression.

21. A method according to paragraph 20, wherein a level comprising a clip operation is added to said compositing table.

22. A method according to any one of paragraphs 18 to 21, wherein a level comprising a push operation is added to said compositing table.

23. A method according to any one of the preceding paragraphs, wherein said compositing table is optimised in regard to the number of pixel operations required to render said image.

24. A method according to any one of the preceding paragraphs, wherein a corresponding compositing expression is a hierarchically structured representation of a particular region represented by said corresponding compositing expression.

25. A method according to paragraph 24, wherein said mapping comprises modifying a manner in which said corresponding compositing expression is evaluated without modifying said hierarchically structured representation.

26. A method according to any one of the preceding paragraphs, wherein said image is at least in part a pixel based image.

27. A method according to any one of the preceding paragraphs, wherein a wholly opaque object in a particular region acts to eliminate one or more operations contributing to at least one other object constituting said particular region, wherein said at least one other object is obscured by said wholly opaque object in a space in which said outlines are defined.

28. A method of optimising an expression representing the layout of one or more objects, each said object having a predetermined outline, said method comprising the steps of:

dividing a space in which said outlines are defined into a plurality of regions, each said region being defined by at least one region outline substantially following at least one of said predetermined outlines or parts thereof;

determining a plurality of further regions depending on at least one characteristic of at least one of said regions, wherein each said further region has a corresponding compositing operation; and

forming said optimised expression from said plurality of further regions and corresponding compositing operations.

29. A method according to paragraph 28, wherein said plurality of further regions modifies a manner in which said expression is evaluated.

30. A method according to paragraph 29, wherein further operators are associated with at least one of said further regions.

31. Apparatus configured to perform the method of any one of the preceding paragraphs.

32. A computer program product including a computer readable medium having a plurality of software modules configured to perform the method of any one of paragraphs 1 to 30.

33. A method of creating an image, said image to be formed by evaluating a hierarchically structured compositing expression representing said image, said hierarchically structured compositing expression consisting of a plurality of graphical objects and operations arranged as sub-expressions, each said object having a predetermined outline, said method comprising the steps of:

determining an active region for each sub-expression of said hierarchically structured compositing expression, said active region being dependent on the predetermined outlines of each graphical object and operations contained in said each sub-expression;

calculating an effective region for a corresponding compositing operation of said hierarchically structured compositing expression, depending on at least one characteristic of at least one active region;

mapping each said effective region and corresponding operation into a compositing table, comprising a plurality of levels, wherein each said level of said compositing table represents at least one operation for rendering an object or part thereof constituting at least one of said effective regions; and

evaluating said hierarchially structured compositing expression using said compositing table.

34. A method of calulating effective regions for a plurality of graphical objects, each said object having a predetermined outline, said method comprising the steps of:

dividing a space in which said outlines are defined into a plurality of active regions, each said active region being defined by at least one region outline following at least one of said predetermined outlines or parts thereof, each said active region having at least one corresponding Boolean expression; and calculating said effective regions depending on at least one charateristic of at least one corresponding Boolean expression, wherein each said effective region has a corresponding Boolean operation.

35. A method according to paragraph 34, wherein said corresponding Boolean operation is a stack operation.

36. A method according to paragraph 34, wherein said corresponding Boolean expression is primitive.

37. A method according to paragraph 34, wherein said corresponding Boolean operation has a primitive left operand.

38. A method according to paragraph 34, wherein said corresponding Boolean operation has a complex left operand.

39. A method of creating an image, said image to be formed by compositing at least a plurality graphical objects, each said object having a predetermined outline, said method comprising the steps of:

dividing a space in which said outlines are defined into a plurality of active regions, each said active region being defined by at least one region outline following at least one of said predetermined outlines or parts thereof;

determining a plurality of effective regions depending on at least one characteristic of at least one active region, wherein each said effective region has a corresponding compositing operation;

mapping said effective regions and corresponding compositing operations into a compositing table, comprising a plurality of levels, wherein each said level of said compositing table represents at least one operation for rendering an object or parts thereof; and

compositing said image using said compositing table.

40. Apparatus configured to perform the method of any one of paragraphs 33 to 39.

41. A computer program product including a computer readable medium having a plurality of software modules configured to perform the method of any one of paragraphs 33 to 39.

42. A method of determining an expression for each of a plurality of active regions substantially as herein described with reference to Table 3 or Table 6.

**Dated 16 March, 1999**
**Canon Kabushiki Kaisha**

**Patent Attorneys for the Applicant/Nominated Person**
**SPRUSON & FERGUSON**

## APPENDIX A

```
/*
 * Compile a compositing expression.
 * Returns a level activation table which evaluates the
 * expression.
 */
Table compile(Expression e)
{
    return compile(e, activeRegion(e));
}

/*
 * Recursive compiler for compositing expressions.
 * Returns a level activation table which evaluates an
 * expression clipped to a given clip region.
 */
Table compile(Expression e, Region clip)
{
    Table table;  // an empty level activation table

    // Handle primitive expresions and right hand leaf nodes.
    if (isPrimitive(e))
    {
        addLevels(table, PUSH, e, clip);
        return table;
    }

    Operation op = operation(e);
    Expression left  = leftOperand(e);
    Expression right = rightOperand(e);

    Region leftActive  = activeRegion(left);
    Region rightActive = activeRegion(right);

    addLevels(table, op,
            intersection(clip,
                    intersection(leftActive, rightActive)) );

    if (isPrimitive(left))
    {
        // Add a level to implement the primary operation
        // combined with the left hand operand
        addLevels(table, op, left,
                                intersect(clip, rightActive));
        // Add an extra push for operations that need it.
        switch (op)
    {
            case OVER:
            case ROVER:
            case OUT:
            case RATOP:
            case XOR:
                addLevels(table, PUSH, left,
                    intersect(clip, complement(rightActive)));
                break;
            default:
                    // Don't do anything.
                    break;
               }
          }
          else
          {
               // Calculate the effective region for the operation.
               Region effective = intersect(clip,
                                   intersect(leftActive, rightActive));

               // Add a level (possibly with duplicates) to evaluate
               // the primary operation.  The level takes its source
               // from the stack.
               addLevels(table, op, POP, effective);

               // Compile the left-hand operand.
               switch (op)
               {
                    case OVER:
                    case ROVER:
                    case OUT:
                    case RATOP:
                    case XOR:
                         append(table, compile(left, clip));
                         break;
                    default:
                         append(table,
                              compile(left,
                                        intersect(clip, rightActive)));
               }
          }
          // Compile the right hand operand, and append its levels to
          // the table.
          switch (op)
          {
               case IN:
               case RIN:
               case OUT:
               case RATOP:
                    append(table, compile(right,
                                        intersect(clip, leftActive)));
                    break;
               default:
                    append(table, compile(right, clip));
          }
          return table;
     }

     /************************************************************/
     /*   Operations on Regions                                   */
     /************************************************************/
     /*
      * Returns false if the region is not empty, but will not
      * always return true if it is.  This is a quick test to
      * eliminate empty regions.
      */
     bool testIfEmpty(Region region);
```

```
/*
 * Returns the number of disjuncts in a Boolean expression.
 */
int numDisjuncts(Region expr);

/*
 * Returns the disjunct at a given index in a Boolean expression.
 */
Region getDisjunct(Region region, int index);

/*
 * Returns the conjunct at a given index in a Boolean expression.
 */
Region getConjunct(Region region, int index);

/*
 * Normalize a boolean expression representing a region.
 * Returns a Boolean expression which is a disjoint union of
 * intersections of primitives or their complements.
 */
Region normalize(Region expr);

/************************************************************/
/*   Operations on Tables                                    */
/************************************************************/
/*
 * Add levels to a level activation table to implement a given
 * operation with a given fill, clipped to a given region.
 * op can be a compositing operation or "PUSH".
 * fill can be a primitive expression or "POP".
 * Duplicate levels will be added if the clip region cannot be
 * represented as a simple intersection.
 */
void addLevels(
              Table & table,
              Operation op,
              Expression fill,
              Region clip)
{
     normalize(clip);
     for (int i = 0; i != numDisjuncts(clip); ++i)
          addClippedLevel(table, op, fill, getDisjunct(clip, i));
}

/*
 * Add a level plus clip levels to a level activation table.
 * The clipping region is expected to be in the form of an
 * intersection of primitive regions or their complements.
 */
void addClippedLevel(
              Table & table,
              Operation op,
              Expression object,
              Region clip)
{
     if (testIfEmpty(clip))  // If empty, don't add a level.
          return;
     Level level(op, fill(object));
```

```
        int index = append(table, level);
        for (int i = 0; i != numConjuncts(clip); ++i)
        {
            Region r = getConjunct(clip, i);
            addClip(table, index, r);
        }
    }

    /*
     * Append a level to a level activation table.
     * Returns the index of the added level.
     */
    int append(Table & table, Level level);

    /*
     * Adds a clip level to a level activation table.
     * If a clip level already exists for the given set of edges,
     * the level to be clipped is added to the existing level.
     * The clip argument must be a primitive region or its complement.
     * If it is a primitive region, the level added will be a CLIPIN.
     * If it is a complement of a primitive, the added level will be
     * a CLIPOUT.
     */
    void addClip(Table & table, int index, Region clip);
```

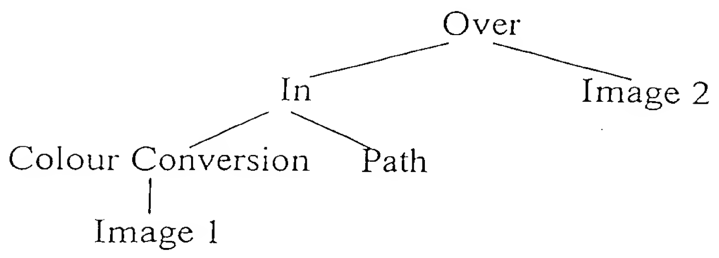


Figure 1.

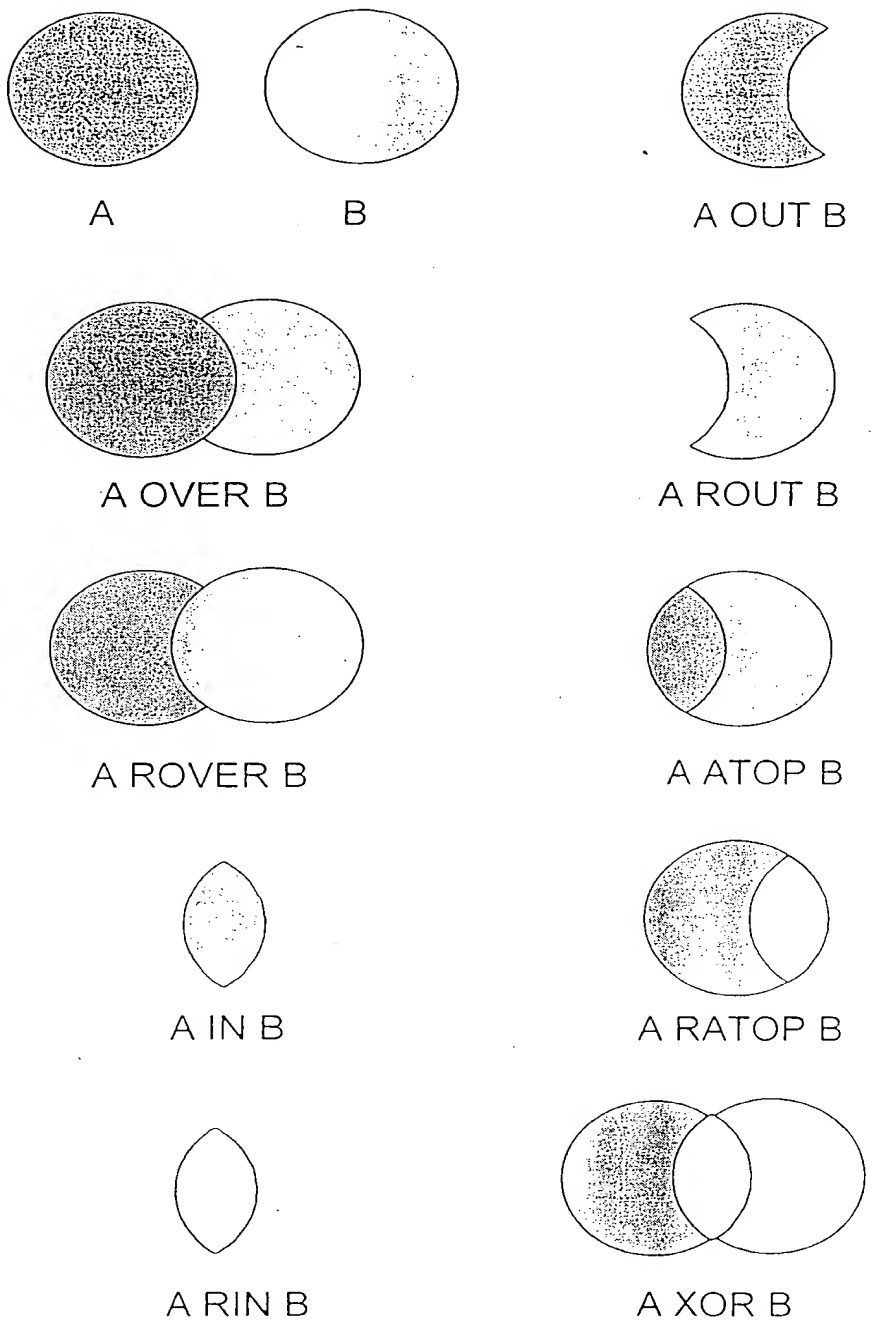

A
B
A OUT B
A OVER B
A ROUT B
A ROVER B
A ATOP B
A IN B
A RATOP B
A RIN B
A XOR B

Fig. 2

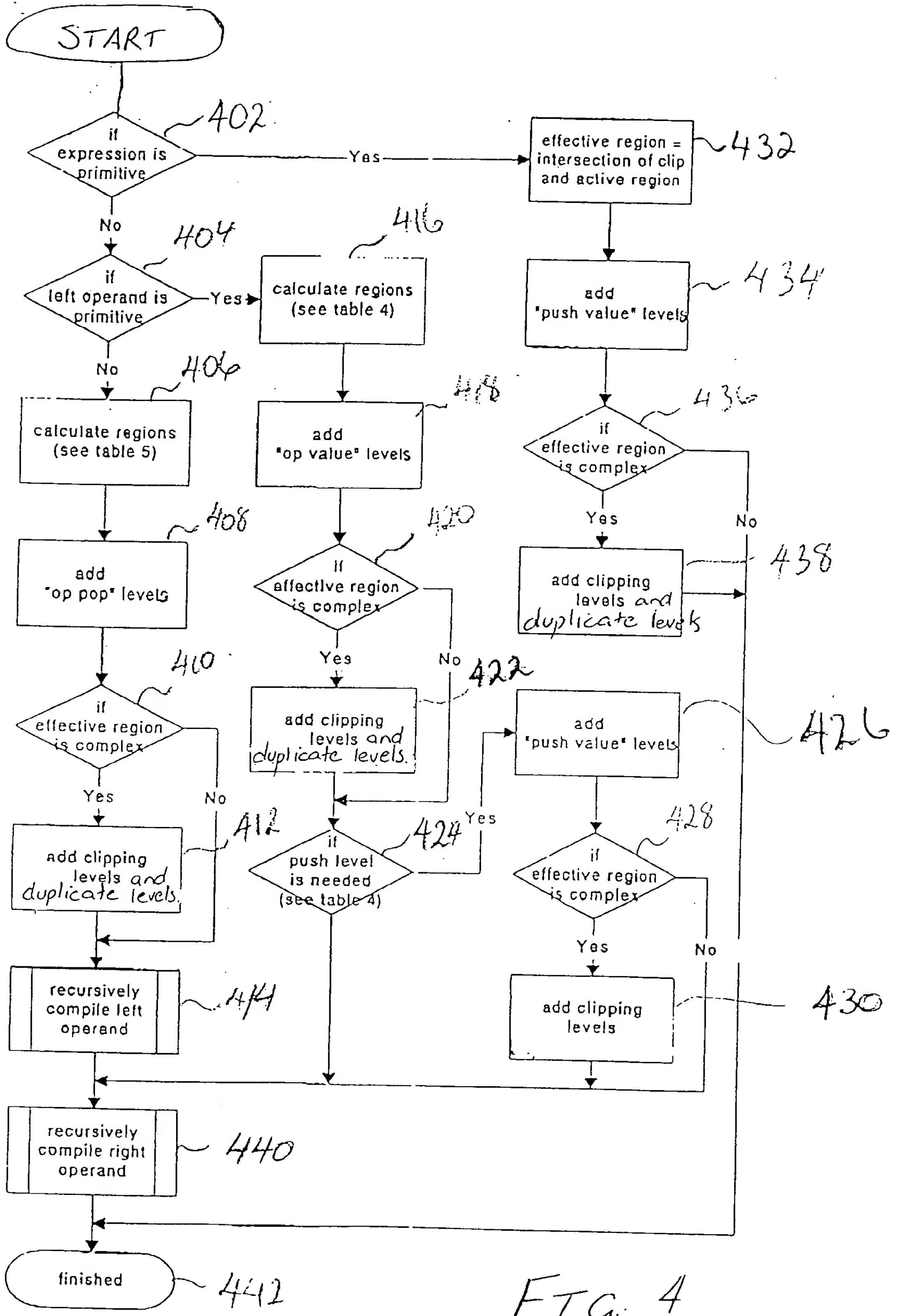
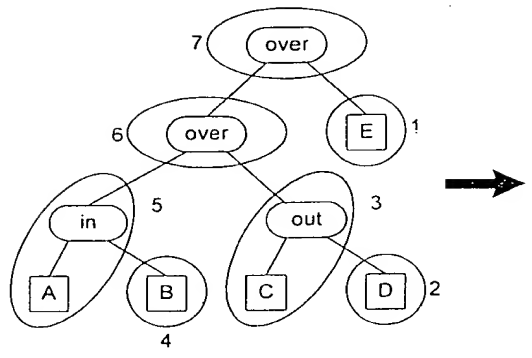

START
402
if expression is primitive
Yes
No
effective region = intersection of clip and active region
432
404
if left operand is primitive
Yes
No
416
calculate regions (see table 4)
434
add "push value" levels
406
calculate regions (see table 5)
418
add "op value" levels
436
if effective region is complex
Yes
No
408
add "op pop" levels
420
if effective region is complex
Yes
No
438
add clipping levels and duplicate levels
410
if effective region is complex
Yes
No
422
add clipping levels and duplicate levels
426
add "push value" levels
412
add clipping levels and duplicate levels
424
if push level is needed (see table 4)
Yes
428
if effective region is complex
Yes
No
recursively compile left operand
414
add clipping levels
430
recursively compile right operand
440
finished
442

FIG. 4



| operation | fill |
|---|---|
| over | pop |
| over | pop |
| in | A |
| push | B |
| out | C |
| push | D |
| push | E |

FIG. 3

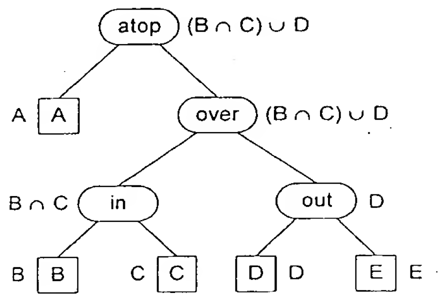


FIG. 5

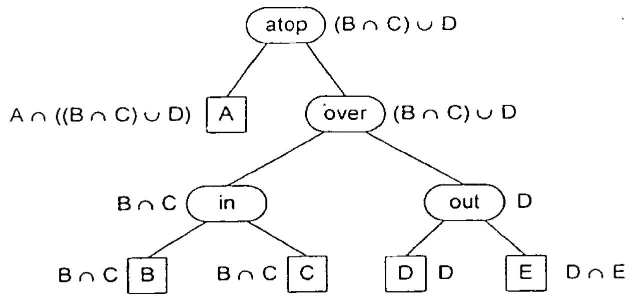


FIG. 6

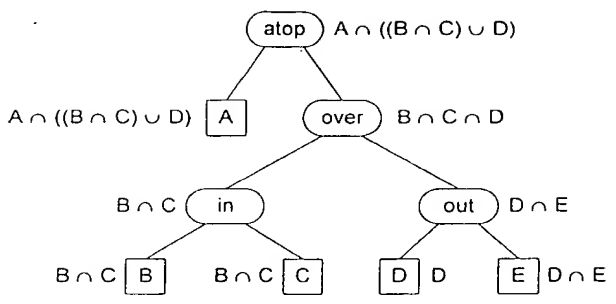
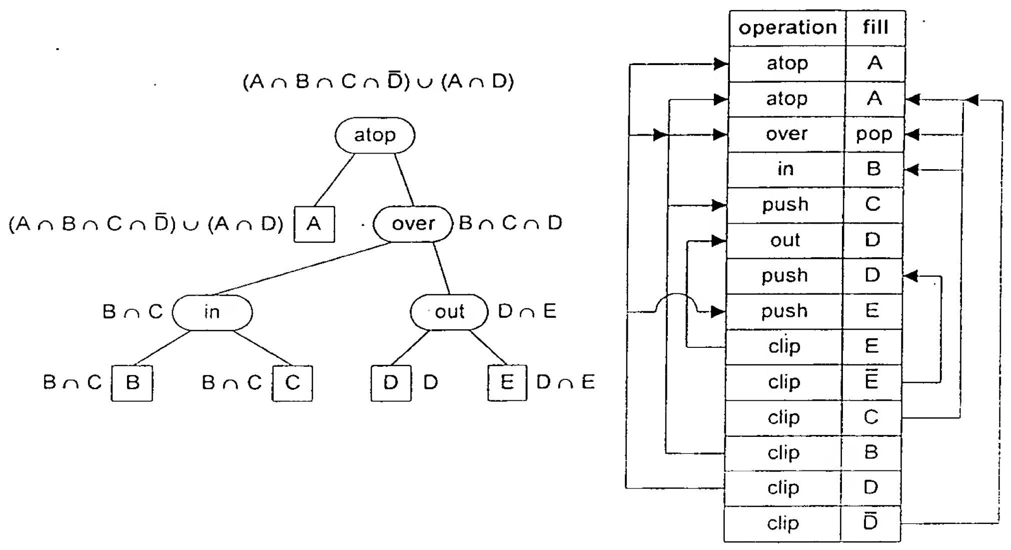


**FIG. 7**



| operation | fill |
|---|---|
| atop | A |
| atop | A |
| over | pop |
| in | B |
| push | C |
| out | D |
| push | D |
| push | E |
| clip | E |
| clip | Ē |
| clip | C |
| clip | B |
| clip | D |
| clip | D̄ |

**FIG. 8**

| Operation | Region | Value | Level | Stack | Result | Correct |
|---|---|---|---|---|---|---|
| A over B | A ∩ B | A over B | over A | B | A over B | A over pop |
| | A ∩ B | A | | ? | A over ? | push A |
| | A ∩ B | B | | B | B | no-op |
| | A ∩ B | glass | | ? | ? | no-op |
| A rover B | A ∩ B | A rover B | rover A | B | A rover B | A rover pop |
| | A ∩ B | A | | ? | A rover ? | push A |
| | A ∩ B | B | | B | B | no-op |
| | A ∩ B | glass | | ? | ? | no-op |
| A in B | A ∩ B | A in B | in A | B | A in B | A in pop |
| | A ∩ B | glass | | ? | A in ? | no-op |
| | A ∩ B | glass | | B | B | pop |
| | A ∩ B | glass | | ? | ? | no-op |
| A rin B | A ∩ B | A rin B | rin A | B | A rin B | A rin pop |
| | A ∩ B | glass | | ? | A rin ? | no-op |
| | A ∩ B | glass | | B | B | pop |
| | A ∩ B | glass | | ? | ? | no-op |
| A out B | A ∩ B | A out B | out A | B | A out B | A out pop |
| | A ∩ B | A | | ? | A out ? | push A |
| | A ∩ B | glass | | B | B | pop |
| | A ∩ B | glass | | ? | ? | no-op |
| A rout B | A ∩ B | A rout B | rout A | B | A rout B | A rout pop |
| | A ∩ B | glass | | ? | A rout ? | no-op |
| | A ∩ B | B | | B | B | no-op |
| | A ∩ B | glass | | ? | ? | no-op |
| A atop B | A ∩ B | A atop B | atop A | B | A atop B | A atop pop |
| | A ∩ B | glass | | ? | A atop ? | no-op |
| | A ∩ B | B | | B | B | no-op |
| | A ∩ B | glass | | ? | ? | no-op |
| A ratop B | A ∩ B | A ratop B | ratop A | B | A ratop B | A ratop pop |
| | A ∩ B | A | | ? | A ratop ? | push A |
| | A ∩ B | glass | | B | B | pop |
| | A ∩ B | glass | | ? | ? | no-op |
| A xor B | A ∩ B | A xor B | xor A | B | A xor B | A xor pop |
| | A ∩ B | A | | ? | A xor ? | push A |
| | A ∩ B | B | | B | B | no-op |
| | A ∩ B | glass | | ? | ? | no-op |

Table 1

| Operation | Region | Value | Level | Stack | Result | Correct |
|---|---|---|---|---|---|---|
| A over B | $A \cap B$ | A over B | over | A,B | A over B | over |
| | $A \cap \bar{B}$ | A | | A,? | A over ? | no-op |
| | $\bar{A} \cap B$ | B | | B,? | B over ? | no-op |
| | $\bar{A} \cap \bar{B}$ | glass | | ?,? | ? over ? | no-op |
| A rover B | $A \cap B$ | A rover B | rover | A,B | A rover B | rover |
| | $A \cap \bar{B}$ | A | | A,? | A rover ? | no-op |
| | $\bar{A} \cap B$ | B | | B,? | B rover ? | no-op |
| | $\bar{A} \cap \bar{B}$ | glass | | ?,? | ? rover ? | no-op |
| A in B | $A \cap B$ | A in B | in | A,B | A in B | in |
| | $A \cap \bar{B}$ | glass | | A,? | A in ? | pop |
| | $\bar{A} \cap B$ | glass | | B,? | B in ? | pop |
| | $\bar{A} \cap \bar{B}$ | glass | | ?,? | ? in ? | no-op |
| A rin B | $A \cap B$ | A rin B | rin | A,B | A rin B | rin |
| | $A \cap \bar{B}$ | glass | | A,? | A rin ? | pop |
| | $\bar{A} \cap B$ | glass | | B,? | B rin ? | pop |
| | $\bar{A} \cap \bar{B}$ | glass | | ?,? | ? rin ? | no-op |
| A out B | $A \cap B$ | A out B | out | A,B | A out B | out |
| | $A \cap \bar{B}$ | A | | A,? | A out ? | no-op |
| | $\bar{A} \cap B$ | glass | | B,? | B out ? | pop |
| | $\bar{A} \cap \bar{B}$ | glass | | ?,? | ? out ? | no-op |
| A rout B | $A \cap B$ | A rout B | rout | A,B | A rout B | rout |
| | $A \cap \bar{B}$ | glass | | A,? | A rout ? | pop |
| | $\bar{A} \cap B$ | B | | B,? | B rout ? | no-op |
| | $\bar{A} \cap \bar{B}$ | glass | | ?,? | ? rout ? | no-op |
| A atop B | $A \cap B$ | A atop B | atop | A,B | A atop B | atop |
| | $A \cap \bar{B}$ | glass | | A,? | A atop ? | pop |
| | $\bar{A} \cap B$ | B | | B,? | B atop ? | no-op |
| | $\bar{A} \cap \bar{B}$ | glass | | ?,? | ? atop ? | no-op |
| A ratop B | $A \cap B$ | A ratop B | ratop | A,B | A ratop B | ratop |
| | $A \cap \bar{B}$ | A | | A,? | A ratop ? | no-op |
| | $\bar{A} \cap B$ | glass | | B,? | B ratop ? | pop |
| | $\bar{A} \cap \bar{B}$ | glass | | ?,? | ? ratop ? | no-op |
| A xor B | $A \cap B$ | A xor B | xor | A,B | A xor B | xor |
| | $A \cap \bar{B}$ | A | | A,? | A xor ? | no-op |
| | $\bar{A} \cap B$ | B | | B,? | B xor ? | no-op |
| | $\bar{A} \cap \bar{B}$ | glass | | ?,? | ? xor ? | no-op |

**Table 2**

| Expression | Active Region |
|---|---|
| A over B | $A \cup B$ |
| A rover B | $A \cup B$ |
| A in B | $A \cap B$ |
| A rin B | $A \cap B$ |
| A out B | A |
| A rout B | B |
| A atop B | B |
| A ratop B | A |
| A xor B | $A \cup B$ |

**Table 3**

| Operation | Levels | Clip |
|---|---|---|
| **A over B** | over A | clip $\cap$ B |
| | push A | clip $\cap$ $\bar{B}$ |
| | table B | clip |
| **A rover B** | rover A | clip $\cap$ B |
| | push A | clip $\cap$ $\bar{B}$ |
| | table B | clip |
| **A in B** | in A | clip $\cap$ B |
| | table B | clip $\cap$ A |
| **A rin B** | rin A | clip $\cap$ B |
| | table B | clip $\cap$ A |
| **A out B** | out A | clip $\cap$ B |
| | push A | clip $\cap$ $\bar{B}$ |
| | table B | clip $\cap$ A |
| **A rout B** | rout A | clip $\cap$ B |
| | table B | clip |
| **A atop B** | atop A | clip $\cap$ B |
| | table B | clip |
| **A ratop B** | ratop A | clip $\cap$ B |
| | push A | clip $\cap$ $\bar{B}$ |
| | table B | clip $\cap$ A |
| **A xor B** | xor A | clip $\cap$ B |
| | push A | clip $\cap$ $\bar{B}$ |
| | table B | clip |

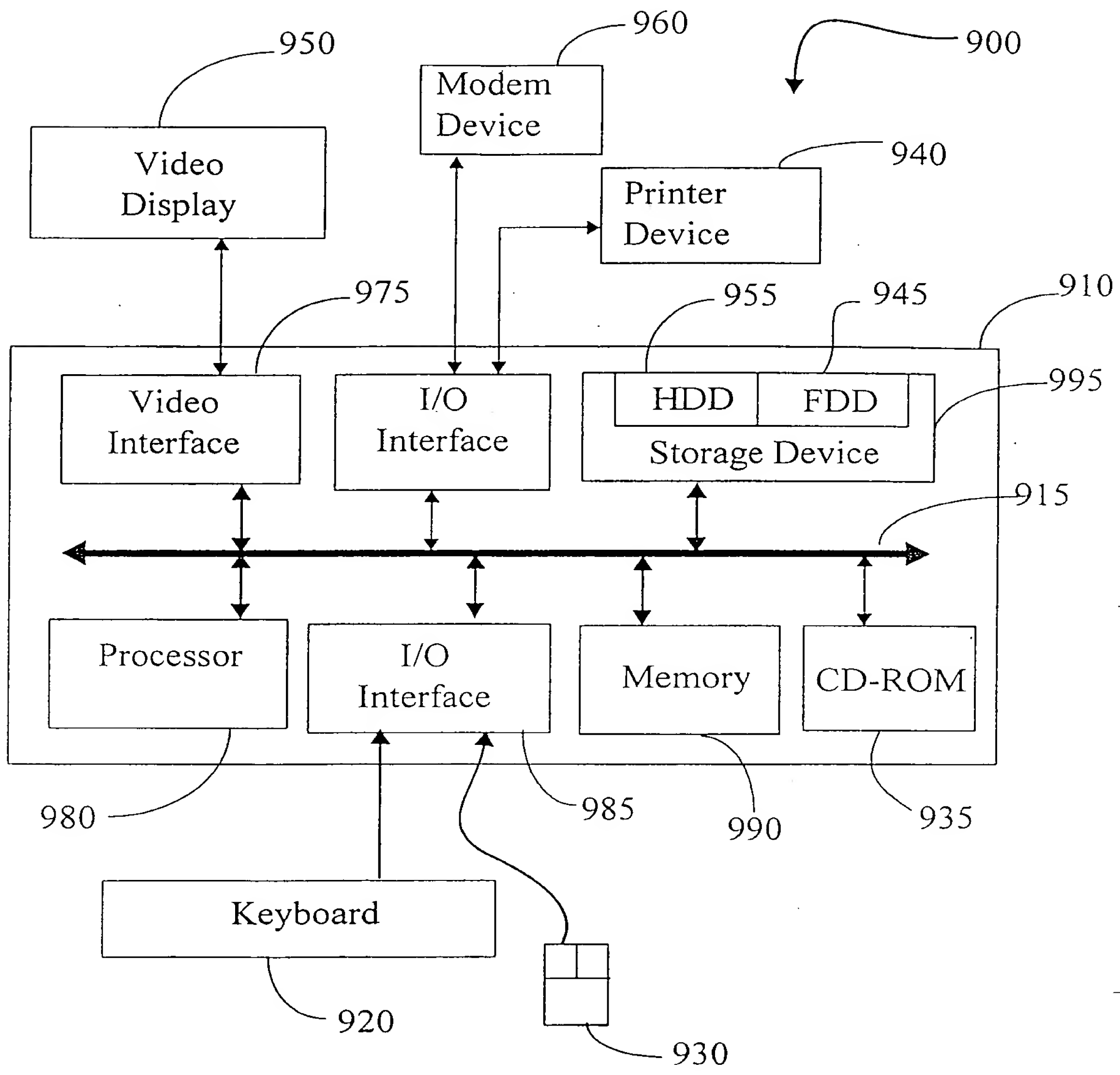
**Table 4**

| Operation | Levels | Clip |
|---|---|---|
| A over B | over | clip ∩ A ∩ B |
| | table A | clip |
| | table B | clip |
| A rover B | rover | clip ∩ A ∩ B |
| | table A | clip |
| | table B | clip |
| A in B | in | clip ∩ A ∩ B |
| | table A | clip ∩ B |
| | table B | clip ∩ A |
| A rin B | rin | clip ∩ A ∩ B |
| | table A | clip ∩ B |
| | table B | clip ∩ A |
| A out B | out | clip ∩ A ∩ B |
| | table A | clip |
| | table B | clip ∩ A |
| A rout B | rout | clip ∩ A ∩ B |
| | table A | clip ∩ B |
| | table B | clip |
| A atop B | atop | clip ∩ A ∩ B |
| | table A | clip ∩ B |
| | table B | clip |
| A ratop B | ratop | clip ∩ A ∩ B |
| | table A | clip |
| | table B | clip ∩ A |
| A xor B | xor | clip ∩ A ∩ B |
| | table A | clip |
| | table B | clip |

**Table 5**

| Expression | Clip A | Clip B | Active Region | Opaque Region |
|---|---|---|---|---|
| A over B | $A$ | $B \cap \bar{O}A$ | $A \cup B$ | $O_A \cup O_B$ |
| A rover B | $A \cap \bar{O}_B$ | $B$ | $A \cup B$ | $O_A \cup O_B$ |
| A in B | $A \cap B$ | $A \cap B \cap \bar{O}_B$ | $A \cap B$ | $O_A \cap O_B$ |
| A rin B | $A \cap B \cap \bar{O}_A$ | $A \cap B$ | $A \cap B$ | $O_A \cap O_B$ |
| A out B | $A \cap B \cap \bar{O}_B$ | $A \cap B$ | $A \cap \bar{O}_B$ | $\bar{B} \cap O_A$ |
| A rout B | $A \cap B$ | $A \cap B \cap \bar{O}_A$ | $B \cap \bar{O}_A$ | $\bar{A} \cap O_B$ |
| A atop B | $A \cap B$ | $A \cap B \cap \bar{O}_A$ | $B$ | $O_B$ |
| A ratop B | $A \cap B \cap \bar{O}_B$ | $A \cap B$ | $A$ | $O_A$ |
| A xor B | $A \cap (\bar{O}_B \cup \bar{O}_A)$ | $B \cap (\bar{O}_B \cup \bar{O}_A)$ | $(A \cup B) \cap (\bar{O}_A \cup \bar{O}_B)$ | $(\bar{B} \cap O_A) \cup (\bar{A} \cap O_B)$ |

**Table 6**



950
Video Display
960
Modem Device
900
940
Printer Device
975
955
945
910
Video Interface
I/O Interface
HDD
FDD
Storage Device
995
915
Processor
I/O Interface
Memory
CD-ROM
980
985
990
935
Keyboard
920
930

FIG. 9